THOR–a display based time sharing system*

by JOHN McCARTHY, DOW BRIAN,
GARY FELDMAN and JOHN ALLEN
Stanford University
Stanford, California

INTRODUCTION

THOR is a time sharing system for the PDP-1 computer with the capacity to run twenty user programs. The system has twenty-eight user consoles, twelve of which are combination keyboard and cathode ray tube display consoles. THOR is designed to capitalize on the display's ability to present large quantities of information quickly and to mitigate the fact that hard copy is not available at display consoles. The other sixteen consoles are Model 33 teletypes with the attendant slow presentation of information and the availability of hard copy.* Because there are more consoles than user programs available, a user program may use more than one console.

THOR was designed to serve a number of purposes:

1. To control the computer-based teaching laboratory. In this application the displays and several other devices including six film chip projectors, a system for presenting audio messages, and teletypes located in schools, are used as teaching consoles. The teaching applications are run as standard time-sharing activities. This has proved quite important to ordinary users and to the teaching laboratory personnel since they require very large amounts of console time for debugging new programs and editing text material.

2. Text editing. Facilities are provided for keeping texts on disk files, creating new files and editing old ones. The texts may represent programs in a variety of languages, teaching material, or any other information organized in pages of lines of characters. Most of the console usage is spent in text editing. The display based text editor has proved itself more convenient and faster than other systems. The lack of immediate hard copy has not proven a serious disadvantage. (Hard copy, when desired, may be obtained by teletype or line printer.)

3. General purpose programming for a small computer. A variety of systems are available including assembly language, an algebraic compiler, an interpreter, a variant of LISP 1.5, and a system for manipulating and displaying functions represented by graphs.

4. To provide time shared access to the IBM 7090 from the display consoles. This has been available to a limited extent at various times. Unfortunately, the 7090 batch processing system has required repeated modifications to give new facilities so the time sharing work has suffered.

The following are the main results of the project:

1. Displays provide a significant improvement over teletypes as time shared consoles. Users decisively prefer them. The large (114) character set and seven character sizes proved valuable.

2. Powerful interactive systems for text editing, on-line debugging and system control have been developed. A flexible system of instructions has been developed which allows the user to design his own interactive systems.

3. It has proved practical to combine a general purpose time-sharing activity with the teaching machine project, a major special use that requires high reliability.

4. Insight has been gained in understanding the nature of tradeoffs in a time sharing system between efficiency, core space, generality, and flexibility.

Details of these matters are given in the following sections.

Hardware

The computer in this system is a Digital Equip-

*Stephen Russell, Brian Tolliver, David Poole, and Paul Stygar also contributed to the work and the paper.

**Display consoles have proved so superior to teletypes that the latter have been retained only as input/output devices in the latest version of the system.

ment Corporation PDP-1, a single-address, 18-bit binary machine.* The central processor has 32 instructions and can address 2^{12} words directly and 2^{16} words indirectly. It has a well developed interrupt system with 16 separate interrupt channels organized in a priority scheme to prevent a lower channel from interrupting a higher priority channel. The input/output connections are easy to modify and inexpensive to extend. This machine lacks an index register and floating point instructions.

There is a restricted mode of operation normally imposed by the system on user programs. In this mode, all attempts by a user program to reference outside of an assigned core area, do input/output, or stop the machine cause interrupts. This lets the system confine the user program space and interpret his input/output commands.

The core memory is attached to two separate controls. The selection of control is made on the high order bit of the address. Each control has connections for four independent devices: the drum, the disk data channel, the display data channel, and the central processor, in descending priority. When idle, each memory control gives its next memory cycle to the highest priority device requesting a cycle from that control. Thus, two of the devices in the system may be getting data from memory at full memory speed simultaneously. For example, the drum may be swapping users from the higher memory while the central processor and the display processor share the lower memory uninterrupted.

Basic to the time sharing system is a very high speed drum whose basic operation is a swap.† In this operation the contents of 2^{12} locations are transferred from core to a drum track, and simultaneously the same core locations are loaded from a different track. This swap takes 33 milliseconds regardless of drum position.

Twelve display consoles serve as the primary user stations. These consoles are capable of displaying 114 different alphanumeric characters, as well as arbitrary line segments (called vectors) and randomly positioned points.

The characters are generated automatically by the display controlled from six-bit binary codes; characters may be displayed in any of seven program selectable sizes. The time required to generate and display one character is only five microseconds. Line segments, or vectors, require five to fifteen microseconds to display. Both characters and vectors may be displayed at any of three intensities. A vector is represented by an 18-bit computer word, which specifies the horizontal and vertical components of that vector. The origin of the vector is ordinarily taken to be the end point of the previously displayed vector. Thus a displayed figure consisting of many line segments may be moved to a different position on the screen by changing only the origin of the first vector in the figure. This ability has proved very useful for programs displaying moving pointers.

The display consoles are all driven and controlled by a single central logic unit. This greatly reduces the total cost of the system, as each console unit can be comparatively simple in design. The information to be displayed on a given console is organized by the program into a table in computer memory. The information in this table will consist of a mixture of control words, six-bit codes specifying characters, and words describing vectors. Approximately once every 30 milliseconds the monitor program starts a data channel which transmits the contents of all the display tables to the display controller, one word at a time. The central logic unit, guided by the control words, displays the information in each table on the proper console. Because the display information is stored in computer memory, small parts of the display can be modified quickly and simply, utilizing the instructions of the main computer. This ability is especially useful in text editing and in other programs requiring rapid visual interaction.

Each display is the size of a small refrigerator with a 16-inch cathode ray tube, and a keyboard mounted at desk height. (See Figure 1 "A Display Console".) The keyboard has 64 keys, with the numbers and letters in a standard typewriter arrangement, with the additional keys at the right side. In addition, there are two control buttons to augment the character

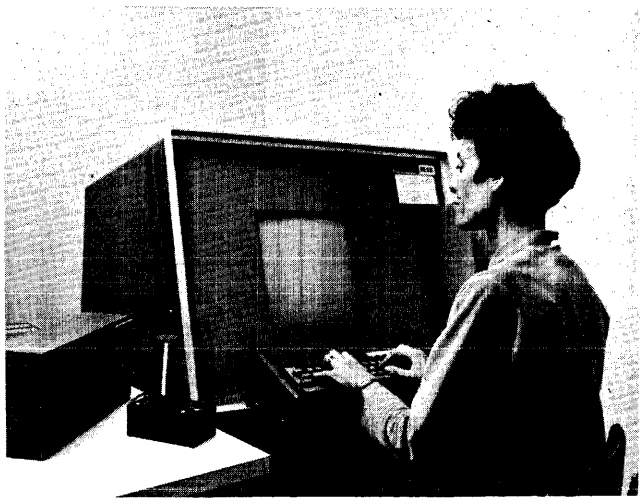

Figure 1 – A display console

*The basic design of this computer was done by the late Benjamin Gurley.

†The idea of the swapping drum for time-sharing is due to E. Fredkin.

code. The keyboards on the displays are logically separated from the rest of the display system, and have a separate interface to the central processor.

When a key is struck an interrupt is sent to the computer and the keyboard locks until the computer reads the keyboard scanner. The computer interrupt program reads the console number, the 6-bit code corresponding to the key, and two bits representing the state of the two control buttons.

Other input/output devices include sixteen teletypes, a data channel to an IBM 7090, an IBM 1301 disk file of 50×10^6 characters, shared with the IBM 7090, and an analog to digital converter. The teaching laboratory consists of six installations each containing a display console, an experimental IBM film-chip projector, and a Westinghouse audio station. The film-chip projector contains 256 microfilm frames which may be projected one or two at a time on an $10'' \times 13''$ screen. There are eight masks underneath the screen which may mask off parts of the image. The projectors can detect a light pen, a feature which allows user interaction.

The audio system is a Westinghouse Prodac-50 computer which controls twelve random access audio tape drives, each of which has 1024 two second messages.

Both the film-chip projector and audio units have maximum access times of two seconds or less.

System configuration

The user is a person who is running a computer program within the THOR time sharing environment. He has a charge number assigned to him which determine his identity for THOR. This number identifies his disk files on a permanent basis; and, when he is using the system, it identifies his console, his individual drum track for storing binary computer programs, and whatever other facilities he may be using at any given time. While the user is logged into the system, he owns at least one console and one drum track. The console consists of a keyboard which allows the user to type information to his user programs and to THOR, and an output device, either CRT display or teletype printer. This output device provides a slate for the user programs and THOR to communicate with the user. When the program is actually being run by THOR it is swapped into PDP-1 core from the user's drum track.

The ordinary user program may occupy 4K of core; however one may request up to 8K additional core. Up to twenty programs may be run by a regular process of bringing a program into core from the drum, allowing it to execute for a short time, marking the state in which its execution is stopped, returning it to the drum and picking up the next user program. User programs are serviced regularly in this fashion on a round robin basis. After a user program has been executed, it is placed last in the queue of user programs waiting to run. Each program in turn is allowed to run for one quantum of time, 64 milliseconds, and then exchanged for the next program. If only one program is in a condition to run it is allowed to run without interruption. The amount of time that the system takes to exchange programs is 33 milliseconds. This swap time is the major source of overhead. However, the swapping time is used to handle system functions and I-O buffering, so it is not entirely wasted.

There are two ways a user can place a binary user program on his drum track. He may prepare an octal program at a console using any of the debugging program such as RAID or the system's octal debugging feature, or he may load a binary image of a program from a previously prepared disk file. Binary files prepared by the assembler and compiler are in the proper format for loading onto a drum track. In addition the system provides a means of saving binary core images from a drum track onto a disk file.

FILES

The main storage device for the system is the IBM 1301 disk. The disk is divided into logical areas called reserved files. Each file is referenced through a unique number and a programmer assigned name. Facilities are provided for creating new files, extending, contacting and destroying old files. The disk file may contain textual information, binary core images, or scratch data in any format. The user may protect his file against unauthorized access.

The system maintains three "internal file numbers" for each user. The user may associate a reserved file with each internal file number. All disk commands then operate in terms of the internal numbers, rather than in terms of the actual name and number of the file. Through this device a user may attach one of his files to an internal file number, then load utility or other programs which operate on his file without having to explicitly open the file for each utility.

With moderate system activity, ten to twelve users, one can expect references to disk file to take no more than 250 milliseconds.

Dsiplays under THOR

We wished to make the display consoles as easy to use as a teletype and yet allow the user access to the full generality of the displays. To this end two display buffers are associated with each of the CRT consoles. One is called the "page printer" buffer and the other is called the "free" buffer. Only one of the

two buffers may be in core at a time; the other is stored on the drum. The user program can control the visibility of these buffers by either executing an instruction which explicitly calls one of the display buffers, or by executing an instruction that implicitly calls a particular buffer.

The "page printer" is used to output characters on the displays in a standard format. When the "page printer" is being used, characters placed in the output buffer of a particular console are displayed under automatic control of the system. As new lines of text are added at the bottom of the display, old lines disappear off the top. A carriage return is automatically inserted whenever a line exceeds the width of the screen, and an * appears at the beginning of the line's continuation. The character, backspace, erases the last character displayed and moves the display pointer back one space. All system messages to the user appear on the page printer. In addition many user programs elect to use this form of output.

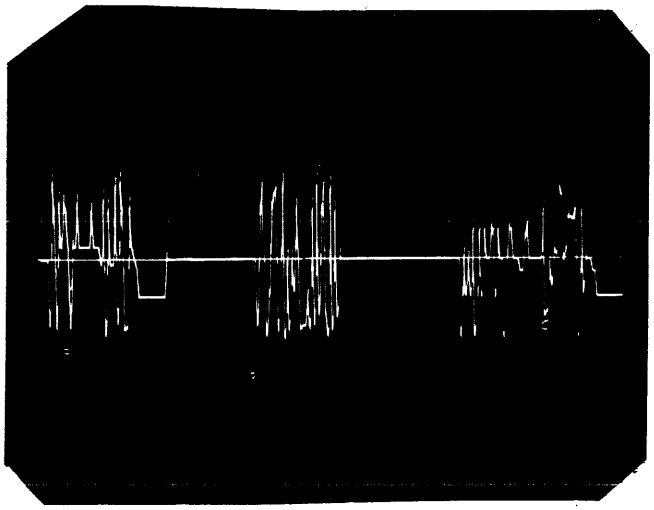

Figure 2 – Display of speech segment
This display was made with vectors deposited in the user's "free" display buffer

The "free" buffer is used for any displays which are more elaborate than the simple page printer. The user is allowed to write display information in the basic language of the display hardware. An example is found in Figure 2 "Display of a Speech Segment." User programs may deposit characters and vectors directly into the "free" buffer. Both the editor and the debugging program described below use this facility.

When the display load is light, a user program may access the buffers which are not in use. This feature allows a program to display on more than one console or to display extra information on a single console. This occasionally proved necessary for display of complex pictures made up of a large number of vectors.

Considering that not every program can make use of the extra core of extra display buffers, on the average less than half the machine is available to each user program. We felt that the wide variety of services that the system supplies and low overhead times possible with fast but voluminous system code are more valuable to the time-sharing user than a little extra space. In particular we felt it vital to keep the time the user waits for response from either his system or his program as low as possible. Wholly new ways of programming arise when response is instantaneous rather than even a few seconds delayed.

In-core section

The in-core section of the THOR system provides services which may be divided into three broad categories:

1. I-O channel routines
2. Scheduling and activation
3. Communication

The I-O channel routines act as the face of THOR turned toward the input/output devices. On an interrupt-priority basis channel routines receive information from all input devices, parcel that information out to the appropriate buffers, and inform the activation section when a user program must be activated to receive its information. Other channel routines accept output from user programs, buffer it, and send it to output devices whenever they are able to receive it.

Activation is handled by a round-robin scheduling algorithm with the exception that programs with disk requests pending are run out of turn to optimize disk usage. The task of the activation routine is to decide whether to remove the current program from core and if so to decide which program is to be run next. A user program may be in one of the following states:

1. **Running.** The user program is in execution. It continues execution until its quantum has expired or until it issues an I-O request that cannot be satisfied within the time remaining in its quantum.
2. **Active.** The user program is ready to run and will be swapped in when its turn comes. A program can become active when:
   a. an output buffer is *almost* empty (this assures continuous output of information),
   b. an input buffer is *almost* full (since it may take some time before the user actually gets to run),
   c. input requested by a user has arrived,
   d. a user determined activation condition has been satisfied (for example, a user program may arrange to be dismissed until a given time arrives),
   e. or when the user commands the system to begin execution of his program.

3. **Waiting.** In this state the program would be active except that it is waiting for the completion of some I-O request or special condition.

4. **Dormant.** The program is not being entered into the round robin. The user may be communicating with the THOR system interpreter or may be dismissed for a variety of user determined reasons.

A program may be swapped out for several reasons.

1. The quantum has expired. The program moves from the running state to the active state.

2. The program has requested the quantum be terminated. The program moves from the running state to the active state.

3. The program has filled an output buffer or has requested input and the input buffer is empty; or the program has requested a special dismissal condition. The program moves from the running state to the waiting state.

4. The program has tried the execute an illegal instruction. The program moves from the running state to the dormant state.

The in-core section provides mechanisms for communication among the users, user programs, and the in-core section. Communication is provided through the input/output transfer instructions (called iot's) which are trapped by the system instructions. When a program executes certain iot's, the system picks up locations in the user program as parameters to service routines. These routines may simulate input/output to the on-line device, control or release ownership of devices, handle character communication, or return information to the user program by filling registers within the program. Through iot's the user program can make its wants known and the in-core section can inform the user program of any variation in the time sharing environment.

Next to services provided by the iot's, character transmission is the major medium of communication. Characters are generated by users typing at keyboards and by user programs sending characters out. Characters go into input buffers to be read by user programs or to output buffers to be printed on scopes or teletype printers. A switchboard provides the possibility for setting up any useful character transmission path: any character source (keyboards and programs) may send to any character sink (input and output buffers). To insure that unwanted connections are not made, facilities are provided so that the user owning any sink may grant or deny permission for a connection into that sink.

This switchboard generality finds application in:

Duplexing—Characters typed at a keyboard can be printed on any console, display or teletype, without user program intervention.

Inter-program-communication—User programs can communicate with each other and with the system interpreter. Thus, several user programs may run as one system coordinating their separate tasks through character communication.

Inter-console-communication—Users at consoles can set up general links for conferences, teaching, monitoring, or chatting.

Multiple-consoles—User programs may receive characters from and send characters to more than one console. This allows a user program to act as a time sharing system within THOR controlling its own set of consoles. Applications include teaching machine monitors and games involving several players.

Character input is a major cause of user program swaps. However, programs vary in how promptly they must pay attention to incoming characters. One limiting case is in the preparation of a file. Here characters should be added to the input buffer when typed, and, when the input buffer gets full, the program should receive all the characters at once and transmit them to the disk. There is no need for this program to be activated each time a character is typed; it need only be activated when the input buffer gets full. On the other extreme is a program whose operation is controlled from the keyboard. Here every character typed should go directly to the program to have an immediate influence on the program's action. To save swaps, we allow each user program to specify under which conditions it should be swapped to receive character input.

In short, the in-core section of the system provides those services which must be performed immediately to allow user programs to continue running with as little delay as possible.

Service programs

There are other services which do not require such fast action. When it is the user rather than the user program who is waiting for the completion of a service, the system need only respond faster than human reaction time. It is of little importance, for example, if the user need wait an extra second to receive an error message. Whenever a service exists whose time of performance only need be faster than human response, that service is given by a user program rather than put within the in-core system. Depending on the nature of the service, the service user program may be given the privileged status of direct access to the in-core part of the system and unrestricted input/output.

There are three types of service program:

The System Interpreter—privileged
Phantoms —privileged

Utilities —not privileged

The System Interpreter acts as the external face of THOR. It accepts commands to THOR from the user typing on his keyboard or from a user program sending characters and communicates to the user by typing into his 'page printer' display buffer or onto his teletype printer. The 'call' character directs all subsequently typed characters to the system interpreter up to and including the next carriage return. This means the user can type requests to the System Interpreter "on the fly" while his user program is running. Because the System Interpreter may be receiving messages from as many as twenty-eight keyboards and twenty user programs it must be run often. Consequently it occupies its private position in the round robin as the twenty-first user being activated whenever there are characters in its input buffer.

A wide variety of services are provided by the System Interpreter:

1. The System Interpreter verifies the user's name and charge number on 'LOGIN' and provides him with a drum track to store user programs. It releases facilities owned by the user on logout.
2. Accounting is based on both the time spent sitting at a console and the time the user program has actually been running.
3. The System Interpreter parcels out such limited facilities as extra consoles, extra display buffers, extra core memory, and input/output devices. Extra consoles may be made 'slaves' so that it is impossible to call the system from them. Slave consoles are used only as character input/output devices. For equipment which several user programs may want to use together, a 'club' is formed with one user as president. Only he has the power to add or delete user programs from the membership list.
4. The System Interpreter provides commands for general file handling and maintenance.
5. The System Interpreter allows the user to save all or part of his binary user program at any time for future use and reference, and to restore it with its state unchanged.
6. The user can start, stop, and continue his user program.
7. The System Interpreter can provide the user with information about the state of his program while it is running, as well as information about the state of in-core tables concerning the user program.
8. The System Interpreter provides commands for calling the various utility programs.
9. The System Interpreter provides a primitive debugging service that allows the user to look at and modify all registers of his core image and look at all the registers of the in-core section of the system.

Phantoms provide a means of charging slow services to the running time of a user program. Phantoms are privileged user programs which run in place of a regular user program in the round robin. Thus, the time that the phantom takes to perform its service for a user program is charged to that user program without degrading the performance of the system for the other user programs. There are two phantoms:

The Error Phantom prints all the error messages for running user programs. Printing a lengthy message may require several quanta; the use of the error phantom 'punishes' the user responsible for the error and no one else.

The Iceberg Phantom handles all modifications to the reserved file directory such as the creation, destruction, lengthening and renaming of disk files. These operations require time-consuming references to file control information on the disk. The Iceberg may be brought into operation by a user program executing certain iot instructions or by the system interpreter acting in the name of a user program.

As an additional refinement, orders for the phantom programs are stacked within the in-core section so that, if a phantom completes a task for a particular user program before the end of its quantum, it may start the next task without additional swaps.

Utility programs are non-privileged user programs which may be called from the disk to perform the workhorse services of the time sharing system. They include a scope text editor, a teletype text editor, assembler, compiler desk calculator, and listers.

Text editor

Virtually all editing of symbolic programs is done on the display consoles, using the TVEDIT text editor. The editor is oriented toward the average user of the system, not just the expert programmer. The central design objective was a simple, easily remembered command structure, which would not require the user to have any knowledge of the manner in which the files are actually stored on the disk.

TVEDIT is a random-access editor interacting with the user on a character-by-character basis. Any change in the text directed by the user is immediately reflected in the display, and the appearance of the display at any time is an accurate picture of the current status of the text file and the editor. An example of displayed text is found in Figure 3 "TVEDIT text for a Demonstration Program."

Both control information and new text are typed from the keyboard. Control characters are distinguished by use of one of the special buttons on the display keyboard. This scheme is felt to be more

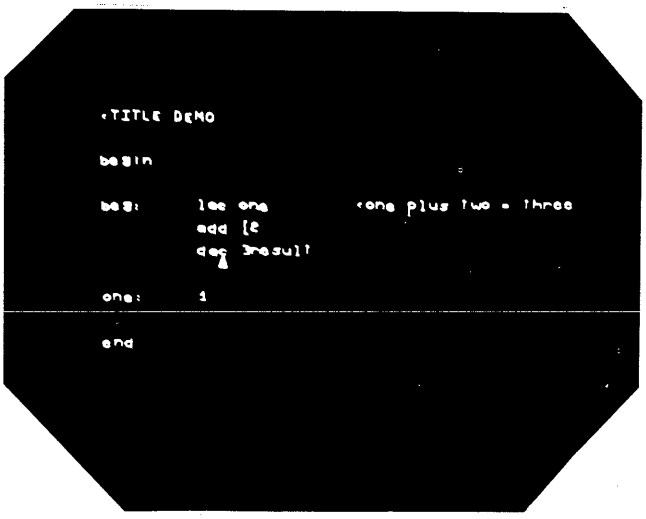

Figure 3—TVEDIT text for a demonstration program
This is assembly text for a simple program to add 1 and 2
The pointer under the 'c' indicates character mode editing

efficient for both the human user and the machine than alternatives involving escape characters or light pens; it allows the carriage return, space bar, and backspace to be used as control characters in a very natural way.

A pointer symbol displayed with the text always indicates the spot at which editing activity will be applied, and also indicates the line/character mode status of the editor. The user can set the pointer to an arbitrary page or move the pointer by lines or characters in any direction. Commands which would move the pointer off of the screen cause a new display "window" to be generated, keeping the pointer in view at all times. Windows usually contain 15 to 20 lines of text. Page divisions in the text are entirely under control of the user, and bear no particular relation to disk records, display windows, or paper sizes.

Following is a very brief description of the complete command set:

n	*space bar*	Move pointer right n characters.
n	*backspace*	Move pointer up n lines or left n characters.
n	*carriage return*	Move pointer down n lines.
n	*K*	Kill (delete) n characters or lines starting at the pointer.
n	*G*	Go to page n.
P		Insert page mark above current line (page marks can be deleted with K).
I		Enter insert mode.
W		Get next display window (allows rapid serial scanning).
F		Finish. Terminates edit run.

The detailed operation of each command is highly context-dependent, with regard to both the current state of the editor and the text being edited. Instead of employing a multiplicity of hard to remember commands, similar functions are lumped into a single command code, and distinctions are made on the basis of factors obvious to the user. For example, "backspace" and "kill" operate by lines or characters, according to the current mode, which is clearly represented in the display. There are no explicit commands for setting line or character modes since this shift is implicit in certain instructions such as spacing into a selected line or entering line mode by using the carriage return. An individual command is limited by the text on which it operates, for example, one cannot space past the end of a line or kill more than a page of text in a single command. The backspace and carriage return normally affect only the pointer and do not change text. After an insert command the backspace deletes preceding characters and carriage returns may be inserted as text. Any command using the control button causes the editor to revert to the normal mode in which typed text replaces existing characters. This type of special-case complexity was deliberately added to make the editor behave in a more natural manner, rather then conform to a set of rigid definitions. Such loosely defined operations are practical only because of the highly interactive nature of the display.

The editor is efficient in usage of machine time, being neither compute-bound nor I/O bound. The average editing run produces a very light load on the system. The random access feature and the file organization greatly reduces the amount of disk activity.

Two hardware factors are worthy of comment in relation to text editing. Efficiency would be enhanced by automatic tab stops in the display equipment; we have to generate a carefully counted series of spaces to achieve presentable tabs. A much more serious problem arises from the basic structure of the character set. Our six-bit characters with separate codes for case shifts generate tremendous problems in all phases of text handling. We cannot emphasize too strongly the importance of a seven-or eight-bit character coding to allow case shift status to be an integral part of each character.

Text files on the disk are organized as a page directory record followed by text records of identical format. This enables the editor to go directly to the

proper record for each new page. Local relative line addressing is easily handled since each record contains forward and backward links to its logical neighbors, and lines within a record are indexed for either forward or backward scanning. Text is packed in serial order in each record, and the link information is hidden behind the end-of-record mark. This fact, together with the unique escape character introducing file control codes, makes it simple for a serial text processor, such as the assembler, to read the file. The explicit "tab" and "carriage return" codes in our character set allows a high density on the file; however, a certain amount of space is normally left free in each record to allow for minor expansions without incurring the cost of linking in an overflow record. This overflow process is invoked automatically when necessary, and the user need not be concerned about such factors.

Random access text editing causes one distinct problem which must be provided for. Upon system or program failure we need to be able to recover as gracefully as possible in the face of lost or improper linkage information. Such a clean-up program is provided, along with serial file read/write services. A merge program is also available for merging selected pages from any number of files.

Assembler

The main characteristics of the assembler were largely determined by conditions of the hardware and software environment in which it operates. The emphasis in the system as a whole on rapid, simple, symbolic debugging dictated that the assembler be as fast as possible, and that symbolic programs be easy to read and modify.

Identifiers may be of any length, with the first six characters unique. Two methods of commenting are provided: one is a single character (<) which causes the rest of the line to be taken as a comment, and the other is the Algol variety, beginning with the word 'comment' and terminated by the next semicolon.

The input format is flexible, involving no fixed columns or fields. Statements are separated by end-of-line, or within the line by semicolons. Spaces, other non-printing characters, and blank lines are ignored. The large character set is used to keep the appearance of the program neat and pleasing, and to avoid confusion by assigning each operation in the assembler to a unique character. Thus, whenever a character appears it has a unique function regardless of context.

Block structure is provided in the assembler for two main reasons. The first is economy of symbol table space. Since each identifier takes at least three 18-bit words, and since the assembler must operate entirely within one 4096 word block, the space recovered by purging local symbols at the end of blocks is very important in assembling large programs. Secondly, block structure facilitates the inclusion of symbolic library routines and the combination of independent programs. The block structure works as in Algol, except that no declarations are required (an important point in machine language code, which tends to have a large number of labels). Prefixing the defining occurrence of a symbol with the character (↑) makes a symbol visible outside the block in which it is defined, allowing subroutines with multiple entry points to be enclosed in a single block.

The requirement for maximum assembly speed suggested a single-pass assembler, and the desire to save symbol storage space with block structure necessitated this approach. The assembler is organized internally as a simple two-stack translator; it evaluates expressions which may contain all of the ordinary arithmetic and logical operators and assembles the values into computer words. In addition there are the usual collection of symbol-defining and assembly control operations, and a general purpose recursive macro processor.

If a symbol is encountered before its definition it is called a forward reference. The symbol is entered in the symbol table, along with the address of the location from which it was referenced. Additional forward references to such a symbol are stored as a linked list in a general storage area shared with the symbol table. As soon as the symbol becomes defined, a "fixup" is issued to every location on the symbol's list of forward references. A "fixup" is a direction to the loader to change the contents of a specified location. References to non-local symbols must be treated as forward references until the end of the block, since no declarations of local symbols are required.

The main symbol table is stored and searched linearly. This facilitates implementation of the block structure, and since most references are to local symbols, searches of the table are usually short. At the end of the block, all symbols local to it are removed from the table, and stored on a disk file for use by the symbolic debugging program.

This assembler is noticeably faster than its predecessors on the same machine, all of which were two-pass processors. It executes between 600 and 900 instructions per word of code assembled, and requires about 75 seconds to assemble itself.

The most interesting conclusions reached in our experience with this processor are that block structure can be very useful in an assembly language, that single-pass assemblers are more efficient than their

multiple-pass counter-parts, and that flexible, readable format of the source program is a great saver of time and frustration.

Debugging

THOR's primary debugging aid is called RAID. It occupies the upper three-eighths of the user's core and is used to monitor the user's program. RAID's display shows the contents of sixteen memory locations in the user's program and the state of his accumulator, in-out register and program flags. The contents of each location are given in both octal and symbolic reconstruction of the assembly text. This presentation is far more informative than the conventional computer console. The program in Figure 3 was assembled and loaded into core. Figure 4, "A RAID Display of a Demonstration Program" shows the result of its execution.

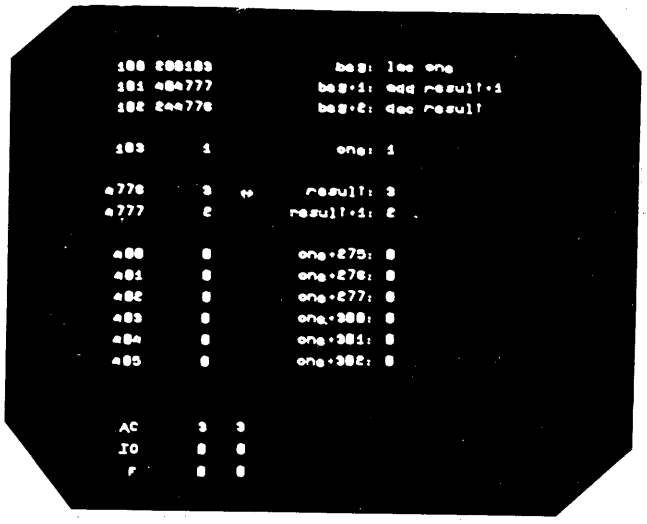

Figure 4—RAID display of a demonstration program
This is a display of the binary and symbolic of the simple demonstration program. The code was executed with the single step feature It left 3 in the accumulator (displayed near the bottom) and in the location 'result' (indicated by the pointer) The array of locations from 'one + 275' to 'one + 302' was displayed to illustrate the number of locations that may be simultaneously visible

RAID displays a pointer next to one of the sixteen locations to act as a focus of the user's attention. By typing single character commands the user can move the pointer to any location on the display. Other commands exist to modify the contents of any location or change the accumulator, in-out register and program flags. Still other commands enable the user to delete old locations and add new locations to the display. The command structure is designed to allow the user to change his focus of attention to the interesting parts of his program in a natural manner. He can trace the program flow, address chains, indirect references, and subroutine calls with a minimum of fuss.

The most useful features of RAID are the variations of single stepping. When the user single steps the instruction at the location indicated by the pointer the instruction is executed and the pointer moves to the next location in the program flow. All displayed locations and registers effected by the execution of that instruction are updated. The user may also plant breakpoints in his code. Normal usage is to plant a breakpoint just before a section of questionable code and start the program. When the breakpoint is reached, control passes to RAID, and all of the displayed locations are updated. The user then single steps through the questionable code, carefully observing the effects of each instruction. As soon as errors appear the user may investigate them immediately.

The single character control language allows skilled users to interact with RAID very rapidly. Typically bursts of such rapid activity will alternate with periods of thoughtful analysis. RAID's value as a self-instructional device is obvious: the novice programmer may enhance his understanding of the various computer instructions by executing them and observing the effects. The experienced programmer may occasionally revamp his understanding of a particular instruction. Generally a programmer single-stepping through his code will encounter occasions on which his image of the situation does not correspond to the actual situation. In writing code a programmer must anticipate the effects of the various instructions and he must maintain an image of what his code does to the memory registers involved. When debugging with RAID, he recreates this anticipated image, and can then correct his thinking where necessary.

The elimination of paper output and lengthy communication with the computer have made a great increase in the effectiveness and speed of debugging.

Usage

Flexible and efficient utility programs and basic system speed have given us a very fast edit—assemble or compile—debug cycle. This has introduced new programming habits. Programmers can now afford to edit and reassemble or compile to purge even moderately trivial bugs rather than make patches to octal code. The practice also helps alleviate the possibility of creating new bugs while correcting old bugs.

Our experience shows that it is easy to live without up-to-the minute listings of programs. One need only know the general position in the text of the proposed changes and a few TVEDIT commands will rapidly locate the desired area.

The fast edit—assemble—debug cycle pays other

dividends. We have observed a tendency toward composing programs on-line. The programmer describes an overview of his program, perhaps sketching out parts of the code, but he leaves the detailed coding for the console session.

THOR may be used to prepare batch processing jobs to be run on the IBM 7090. One method consists of writing the program using TVEDIT, and then converting the text to a disk file format compatible with the IBM processors. To use programs one submits a short job into the 7090 batch processing queue which calls the program from the disk.

Information may also be sent directly between the PDP-1 and the 7090 through a direct data channel. The PDP-1 interrupts the 7090 batch processor between jobs in a process known as time-stealing. Consequently, THOR users may prepare a TVEDIT image of the 7090 job, convert it to the BCD character code, and send it through the data channel. The 7090 sends output back which may be displayed immediately or placed in a disk file to be examined later.

A good example of a user program employing many of the features of THOR is our implementation of the Culler-Fried functional analysis display system. The left special control button was used to distinguish characters standing for operations from those standing for functions. Most operators and all functions are stored on the disk. Rapid disk access was critical in making the system practical.

The structure of the system was sketched out, but the majority of the code was composed at the console. About 4000 words of code were written and debugged in less than two man-weeks. A THOR display console was in use six to eight hours a day and the edit-assemble-debug cycle was constantly exercised. We feel that the Culler System would have taken at least twice as long to develop at a teletype console and months in a batch processing system. It has been impossible to gain exact statistics on the virtue of displays versus teletypes since no users could be coerced into using teletypes if display consoles were available. It seems that the teletype versions of most utility programs are harder to learn, less general, and more difficult to use.

The benefits of efficient and forgiving system and command languages cannot be overemphasized. If a time sharing system is to be used as a good debugging tool, the user must be able to spend long hours at a console without feeling frustration due to excessive waits or errors caused by either himself or the system.

The in-core system, system interpreter, and the error phantom can all be modified while the system is in operation. Thus partial system failures do not necessitate stopping the system. Naturally the normal THOR user may not be so omnipotent but by proper setting of the PDP-1 console test word switches any user program can attain privileged status. The system interpreter and error phantom were written and debugged as user programs with occasional sorties into privileged mode. Though a certain amount of care and caution must be exercised, this feature has proved invaluable.

Computer based teaching laboratory

One of the major projects under THOR has been a system of programs designed to teach mathematics and reading to elementary school children. The following is a brief outline of one of the programs, a drill program, and its use of THOR features:

The drill program gives practice in arithmetic and spelling skills. Twelve teletype consoles are located in local elementary schools. Each of the three hundred children in the experiment does twenty to thirty problems on the console in less than three minutes. Average response times are one to six seconds, so a fast system response time is necessary to keep up the pace. The consoles are placed in slave status to prevent the children from stopping the drills by calling the system interpreter.

The drill program is actually run by three user programs. One is a monitor which keeps a log of usage. The second is an elaborate report generator which may be called while a child is typing so that the teachers receive immediate data analysis of his progress and errors. The third program handles the typing of the problems on the teletypes, the receiving of answers, and data recording. The programs communicate with each other through the character switchboard, the extra user core memory, extra drum tracks available to user programs, and disk files modified while the programs are running. In short the drill programs make full use of the generality provided by THOR for program and console interaction.

REFERENCES

1 J McCARTHY et al
A time-sharing debugging system for a small computer
Spring Joint Computer Conference pp 51-57 1963

2 A KOTOK
DEC debugging tape
Memo MIT-1 rev MIT Cambridge Mass December 11 1961

3 J SAUTER
The Stanford University PDP-1 manual
Stanford Time-Sharing Memo No 36 August 30 1965

4 B TOLLIVER
TVEDIT
Stanford Time-Sharing Memo No 32 March 1 1965

5 P STYGAR
RAID
Stanford Time-Sharing Memo No 37 Nov 2 1965

6 B W LAMPSON
Interactive machine language programming
Fall Joint Computer Conference pp 473-481 1965

7 B D FRIED
The STL On-Line Computer
Vols 1 and 2

8 G J CULLER
Function Oriented On-Line Analysis
Workshop on Computer Organization pp 191-213 1962

9 J GILMORE
Lincoln Lab memo out of print

10 G STRACHEY
Time-sharing in large fast computers
in Proceedings of the International Conference on Information Processing pp 336-341 UNESCO Paris 15-20 June 1959 UNESCO Paris 1960